Jaden Peacock

CMPEN 454 Project 2

Results for task 3.1

In task 3.1, my code essentially was just declaring a lot of the variables that's going to be used and displaying the values of the K_mat, P_mat, R_mat, and position of the images.

Which are the internal parameters? The internal parameters are foclen, prinpoint, aspectratio, and skew.

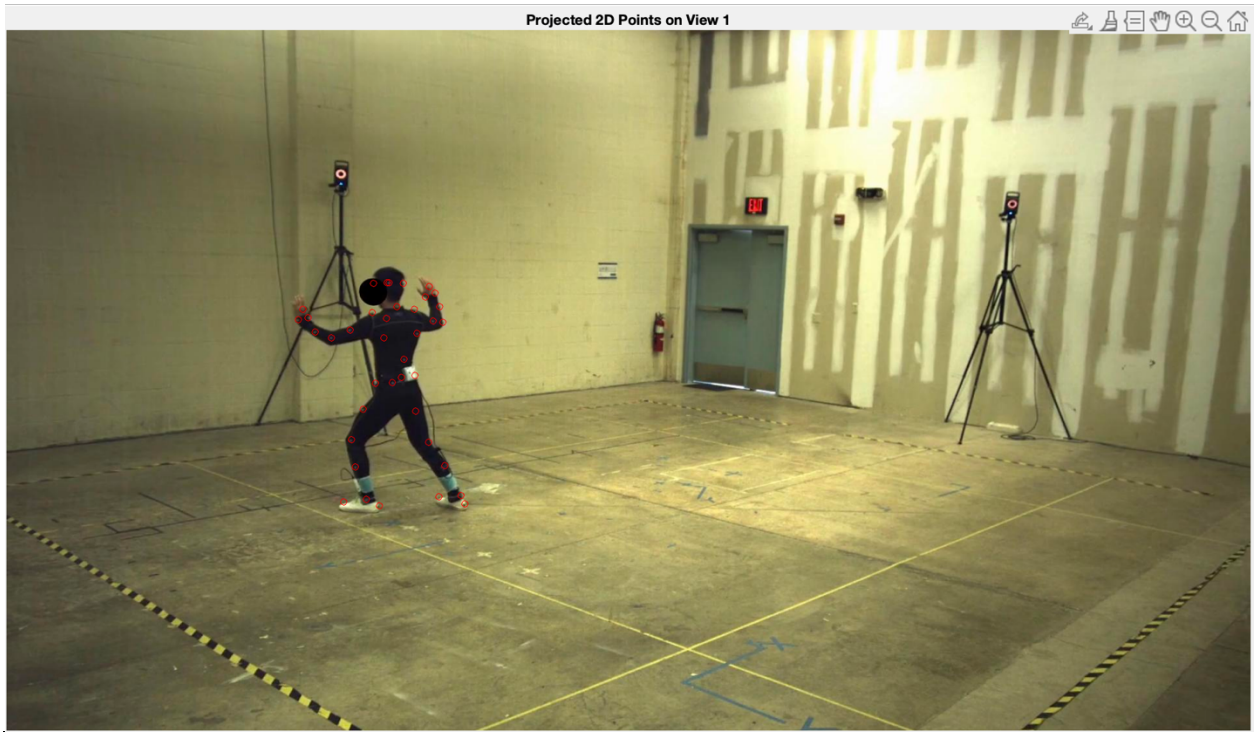
Which are the external parameters? The external parameters are position and Rmat

Which internal parameters combine to form the matrix Kmat? Foclen, prinpoint, aspectratio, and skew are combined to form Kmat

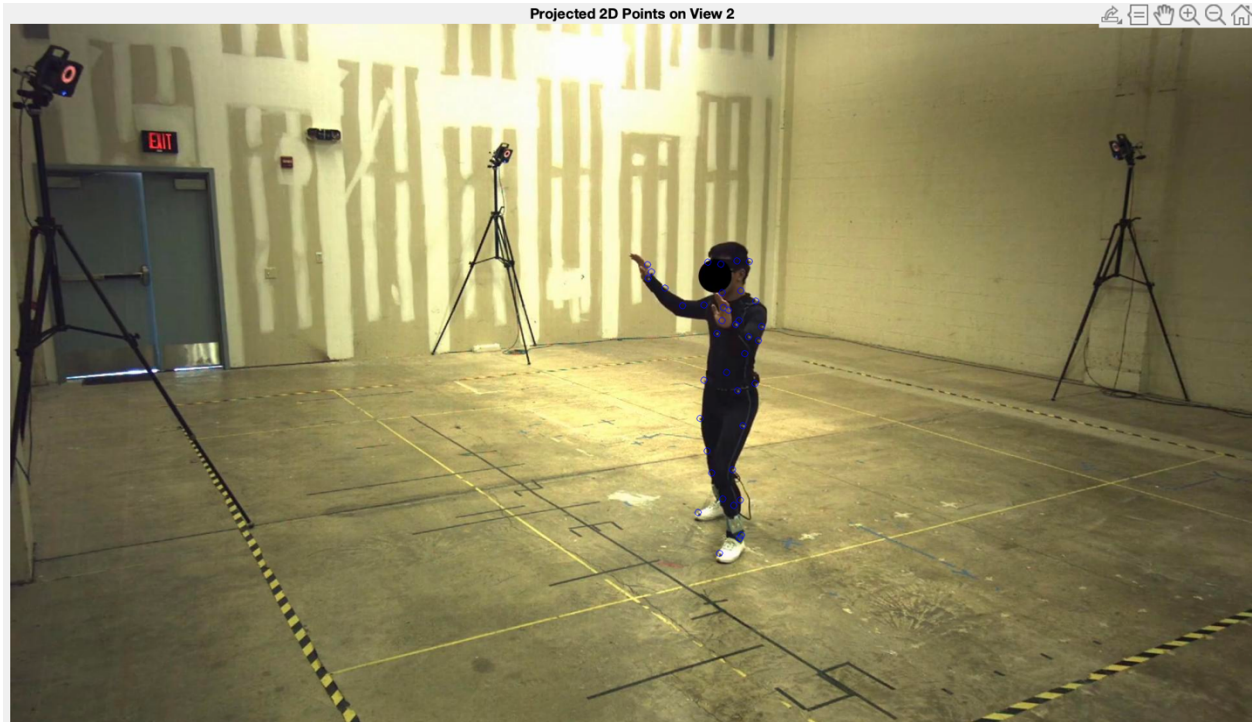
Which external parameters combine to form the matrix Pmat? It combines K (intrinsic matrix), Rmat, and translation vector or T = -Rmat * C

What is the location of the camera? The location of the camera in figure 1 is -4.4630, 5.5455, 1.8811. The location in figure 2 is 4.3906, 5.5110, 1.7887

Results for task 3.2



This is the picture of the points for image 1



This is this the image of the points for image 2

For task 3.2 I originally didn't change the units of the pts3D' and it led to image 2 points not being on the person, I needed to divide mocap by 1000 to turn the units to pixels. For the translation vector, I needed to multiply the negative rotation vector by the transposed position vector for both images. I created a project3Dto2D function to be able to turn the 3D mocap points into 2D pixel points.

The illustrations for task 3.2 are very close to what they should be. When the images come up all the points are overlapping the person's body.

Results for task 3.3

For task 3.3 I first computed the projection matrices for each image using the parameters given and calculated. I put those points into a function called triangulatePoints that converted homogenous coordinates into 3D coordinates. I calculated the mean squared error between the computed projection matrices and the given 3D matrix to see if there were any discrepancies.

My mean squared error was extremely small. It was 9.9198e-31.

Results for task 3.4

For task 3.4, I used a combination of the triangulatePoints function and Plane function to find the coordinates of the planes and items we needed to. The plane function would fit a set of 3D points into the equation ax + by + cz + d = 0.

For the floor coordinates I am consistently getting Z below one but can't get exactly Z = 0.

For the equation of the wall, I am getting .0082x + .1894y + .0256z + .9815 = 0

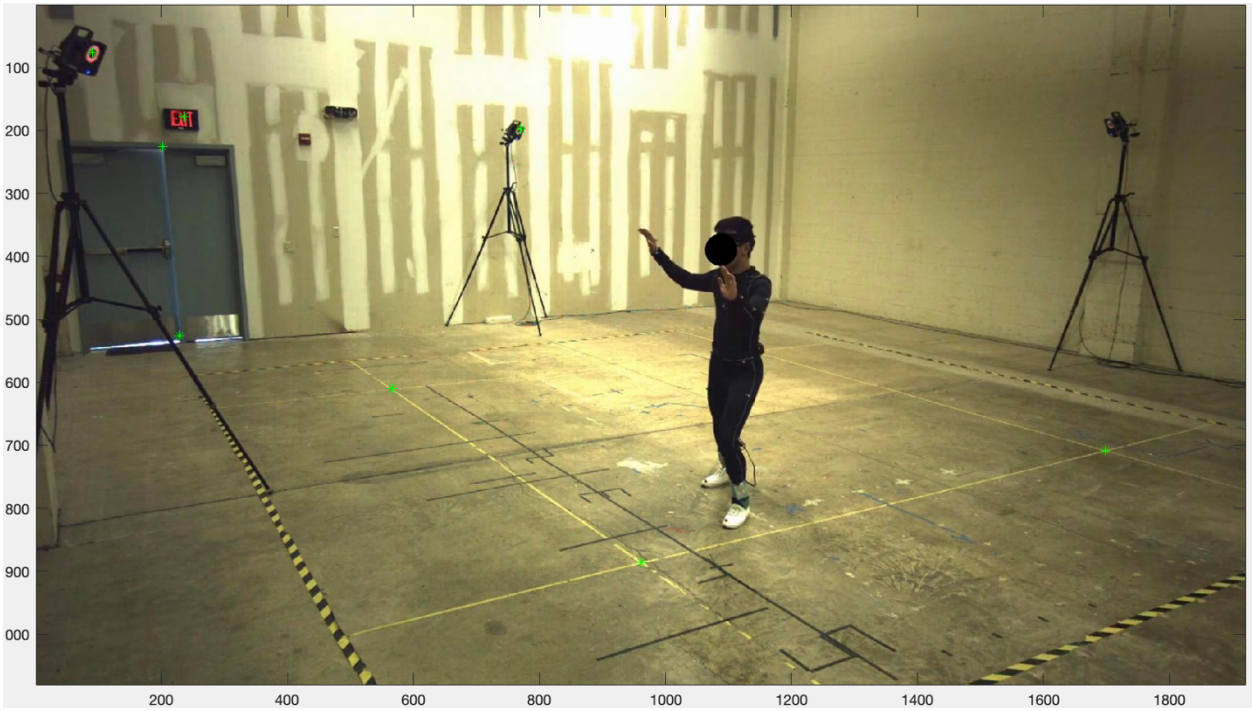
For the Doorway I am consistently getting around 2 units

For the height of the person, I am getting around 1.6321 units

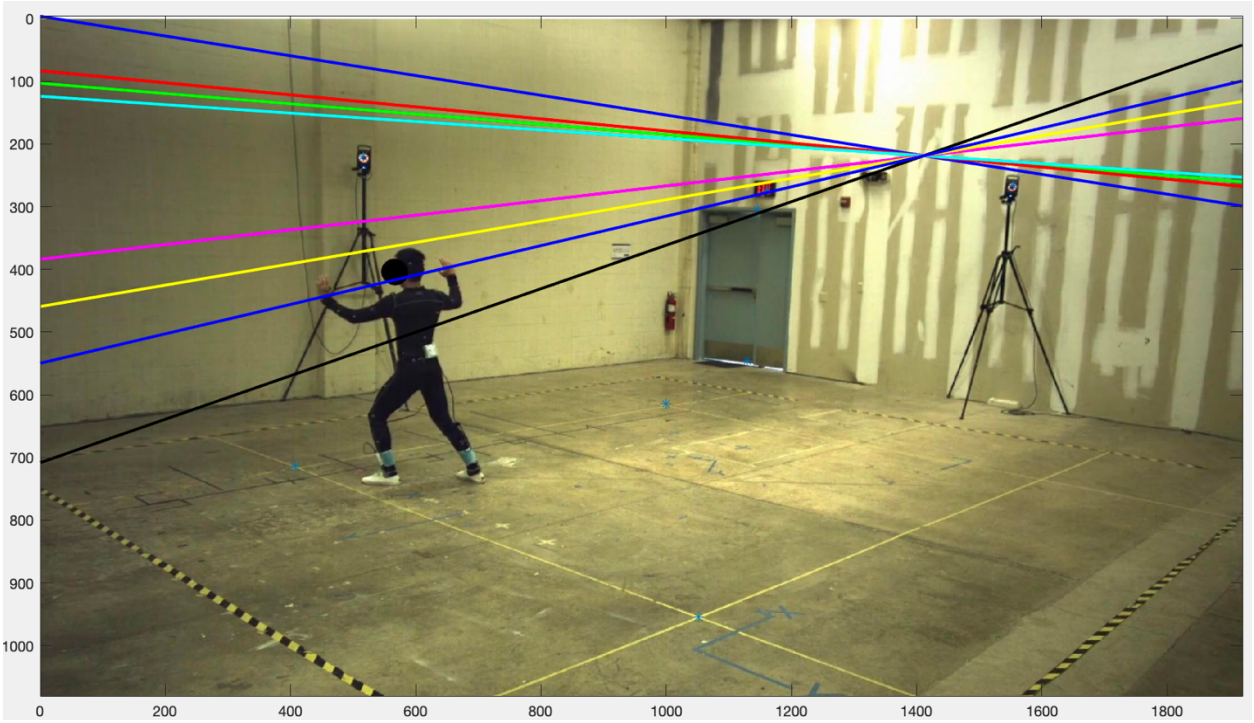
For the camera I am getting [-.0559, -4.9204, 2.4246]

Results for task 3.5

For task 3.5 a function called FundamentalMatrix was used. In this function the process was to find the rotation from camera1 to camera2 by multiplying the rmat of camera 2 to the transpose of camera1. That result was then multiplied by the position of camera 2 minus camera 1 to create the translation. That translation was used to determine values of a skew symmetric matrix, which was used to compute the essential matrix, which was finally converted to the fundamental matrix using the intrinsic matrices of both images.



This is a visual representation of the points I used for task 3.5, for the sanity check, I allowed the user to choose 8 points for both images. The image above is the points chosen on image 2.



These are the epipolar lines created from combining the F_5 matrix with 8 chosen points on image 1.

This is the matrix that was ended up being calculated by my function for the fundamental matrix: -0.0000 -0.0000 0.0013

-0.0000 0.0000 -0.0062

0.0000 0.0052 -0.4205

Results for task 3.6

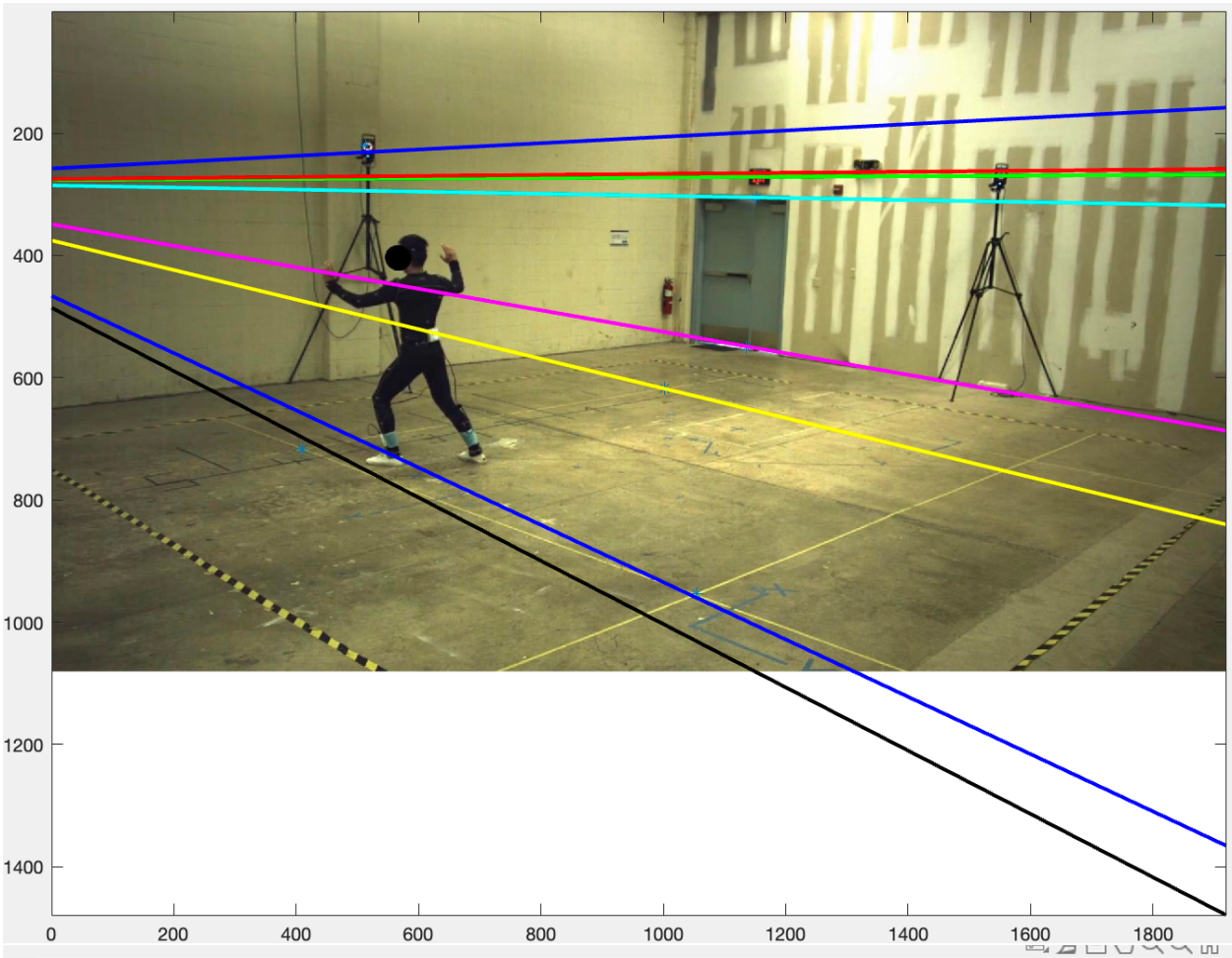
For task 3.6, majority of the code for it was the sample code given to us in class for the eightpoint function. The user is allowed to choose 8 points, and they should correspond between the two images. This is the matrix for the normalized:

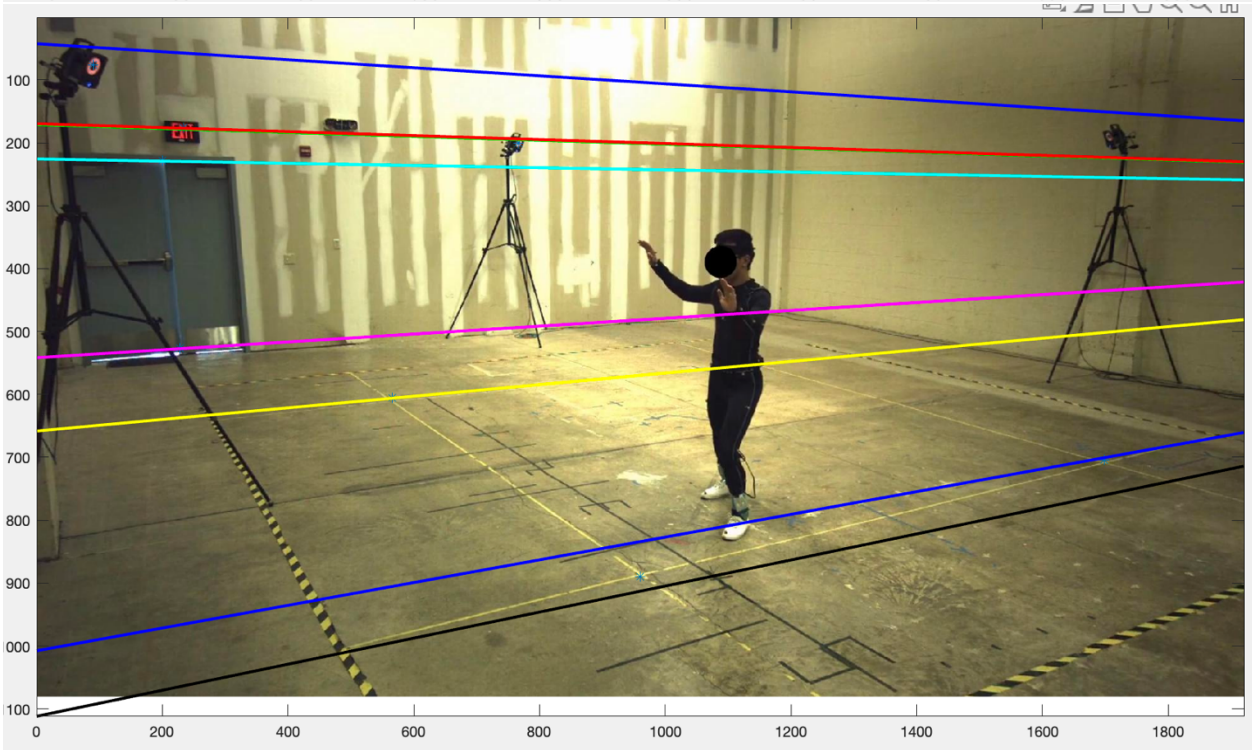
0.0000 -0.0000 0.0002

-0.0000 -0.0000 -0.0003

0.0002 0.0027 -0.6521

This is the epipolar images that I got when trying my best to choose the same point for both images.

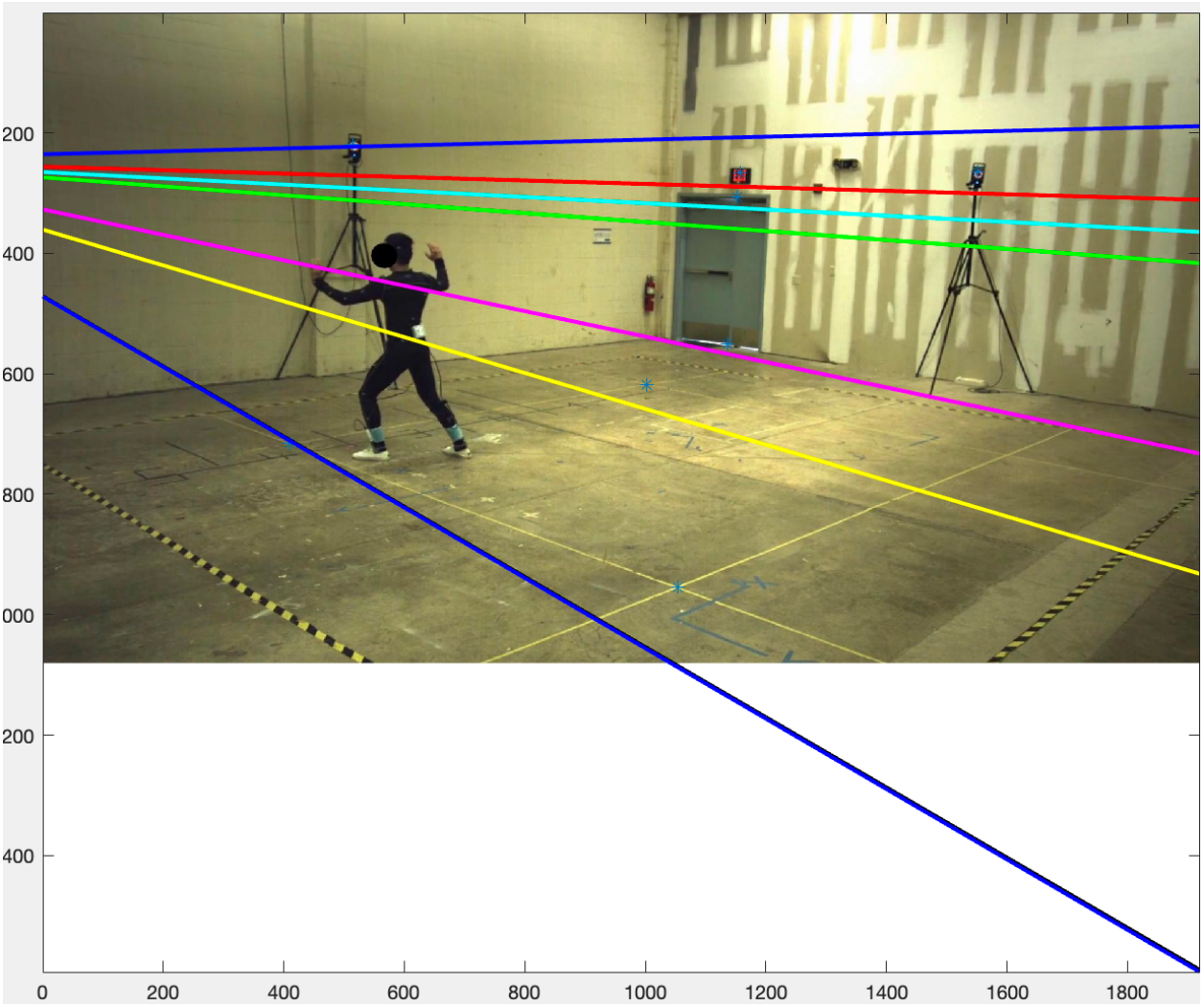

200
400
600
800
1000
1200
1400
0
200
400
600
800
1000
1200
1400
1600
1800

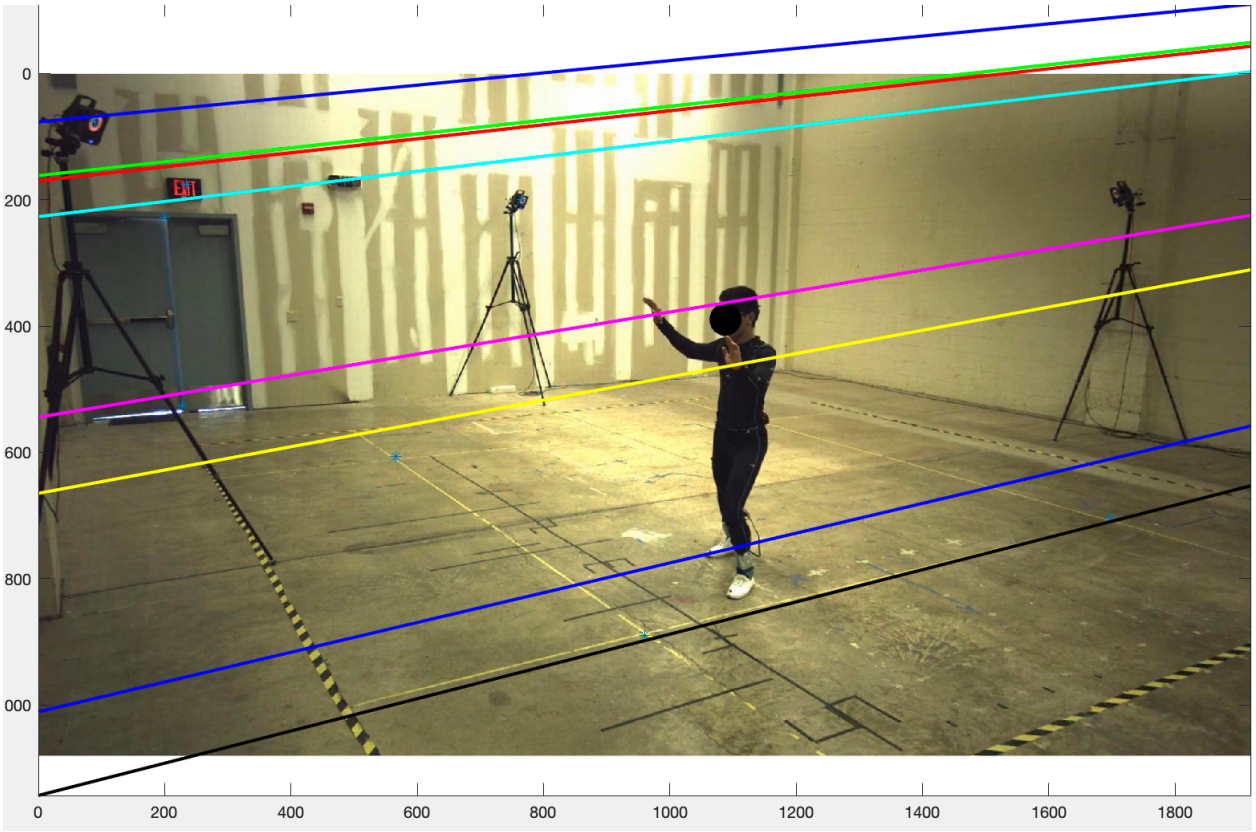

100
200
300
400
500
600
700
800
900
000
100
0
200
400
600
800
1000
1200
1400
1600
1800

To create the non-normalized matrices we were asked for, I used the same sample code from class, but I removed the Hartley normalization used in the function to create a new function. This is the matrix for the non-normalized:

0.0000 0.0000 -0.0001
  0.0000 0.0000 0.0005
 -0.0003 -0.0045 1.0000

This is the epipolar images for the non-normalized image.





The epipolar geometry for both do look reasonable but the normalized version is much better than the non-normalized version when it comes to being closer to the points in the image.

Results for task 3.7

SED for F1 (from camera calibration): 2404111.5802
SED for F2 (eight-point algorithm without normalization): 7359.9982
SED for F3 (eight-point algorithm with Hartley normalization): 220.8187

For task 3.7 I couldn't get the camera calibration to be lower than the normalized and non-normalized values, somewhere within the math I did it wrong and it caused the value to be gigantic. For task 3.7 I created a function called SymmetricEpipolarDistance where it would go through all of the epipolar lines for the images and how far they were from where the point was. I then finally did the total distance / number of points to find the mean. Comparing the Hartley normalization and the Non-normalized, the Hartley normalization was much better by a large margin.